A Comparison of Approaches for Solving Hard Graph-Theoretic Problems

Victoria Horan [*]
Air Force Research Laboratory
Information Directorate

Steve Adachi [†]
Lockheed Martin

Stanley Bak [‡]
Air Force Research Laboratory
Information Directorate

May 1, 2015

Abstract

In order to formulate mathematical conjectures likely to be true, a number of base cases must be determined. However, many combinatorial problems are NP-hard and the computational complexity makes this research approach difficult using a standard brute force approach on a typical computer. One sample problem explored is that of finding a minimum identifying code. To work around the computational issues, a variety of methods are explored and consist of a parallel computing approach using Matlab, a quantum annealing approach using the D-Wave computer, and lastly using satisfiability modulo theory (SMT) and corresponding SMT solvers. Each of these methods requires the problem to be formulated in a unique manner. In this paper, we address the challenges of computing solutions to this NP-hard problem with respect to each of these methods.

1 Problem Statement and Background

First introduced in 1998 [4], an *identifying code* for a graph G is a subset of the vertices, $S \subseteq V(G)$, such that for each $v \in V(G)$ the subset of vertices

[*]`victoria.horan.1@us.af.mil`, Corresponding Author
[†]`steven.h.adachi@lmco.com`
[‡]`stanley.bak.1@us.af.mil`
Approved for public release; distribution unlimited: 88ABW-2015-2163, DIS201503003.

Report Documentation Page		*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **01 MAY 2015**	2. REPORT TYPE	3. DATES COVERED **00-00-2015 to 00-00-2015**
4. TITLE AND SUBTITLE **A Comparison of Approaches for Solving Hard Graph-Theoretic Problems**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Air Force Research Laboratory,,Information Directorate,, , ,**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
In order to formulate mathematical conjectures likely to be true, a number of base cases must be determined. However, many combinato- rial problems are NP-hard and the computational complexity makes this research approach di cult using a standard brute force approach on a typ- ical computer. One sample problem explored is that of nding a minimum identifying code. To work around the computational issues, a variety of methods are explored and consist of a parallel computing approach using Matlab, a quantum annealing approach using the D-Wave computer, and lastly using satis ability modulo theory (SMT) and corresponding SMT solvers. Each of these methods requires the problem to be formulated in a unique manner. In this paper, we address the challenges of computing solutions to this NP-hard problem with respect to each of these methods.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**24**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

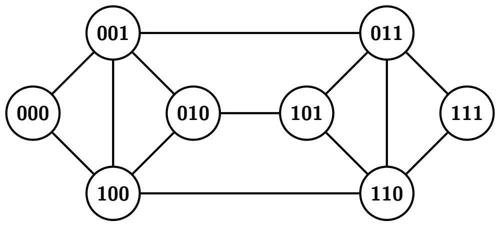

Figure 1: The 2-ary de Bruijn graph of order 3, or $\mathcal{B}(2,3)$.

of S that are adjacent to v is non-empty and unique. That is, each vertex of the graph is uniquely identifiable by the non-empty subset of vertices of S to which it is adjacent. More formally, let $N(v)$ be the set of vertices adjacent to v and $B(v) = N(v) \cup \{v\}$. Then we require that any two vertices $u, v \in V(G)$ have different identifying sets, or more precisely that we must have $S \cap B(v) \neq S \cap B(u)$, and also that both $S \cap B(v), S \cap B(u) \neq \emptyset$. The combinatorial problem of finding minimum identifying codes has been shown to be NP-complete [3], but also has many potential real-world applications.

As some NP-complete problems are notorious for having special graph classes on which there are simple solutions, previous research has focused on the class of de Bruijn networks [2]. This paper explores the problem of finding the minimum size of an identifying code over the undirected de Bruijn graph using three different methods. Section 2 explores a parallel computing algorithm, Section 3 discusses mapping the problem onto the D-Wave computer and using a quantum annealing approach, and finally Section 4 illustrates the method of using Satisfiability Modulo Theory (SMT) solvers.

While many of our examples and data revolve around the class of de Bruijn graphs, the methods discussed throughout can easily be applied to arbitrary graphs. For reference, we provide some of the basic definitions and background here. The undirected d-ary de Bruijn graph of order n, denoted $\mathcal{B}(d,n)$, is the graph with the following vertex and edge sets.

$$\begin{aligned} V &= \{x_1x_2\dots x_n \mid x_i \in \{0,1,\dots,d-1\}\} \\ E &= \{(x,y) \mid x_2x_3\dots x_n = y_1y_2\dots y_{n-1} \text{ or } x_1x_2\dots x_{n-1} = y_2y_3\dots y_n\} \end{aligned}$$

For example, the graph $\mathcal{B}(2,3)$ is illustrated in Figure 1. These graphs have many useful properties for applications, such as having a relatively high number of nodes, a low degree at each node, and many short paths between any two nodes.

2 Parallel Computing

The Information Directorate of the Air Force Research Laboratory is home to one of the Department of Defense's high-performance computing systems

(HPC). Installed on the HPC is Matlab, including the Parallel Computing Toolbox. The pseudocode in Algorithm 1 describes our brute force algorithm, implemented in Matlab.

Algorithm 1 Brute Force Algorithm

```
procedure BruteForce($d, n, k$)                    ▷ $\mathcal{B}(d,n)$ and subset size $k$
    Create list of all subsets of $k$ nodes
    for $i = 1 : \binom{d^n}{k}$ do
        if subset $i$ is a valid identifying code then
            Display subset $i$ to user
        else
            Do nothing
        end if
    end for
end procedure
```

Parallelizing our algorithm takes two steps. The first step is to replace "For" on line 3 with "Parfor". This indicates to Matlab to use the parallel computing toolbox and run each loop iteration independently. The second step requires moving the construction of subsets inside the Parfor loop. Because of the exponential increase in the number of subsets created, it is more efficient to generate each subset within the loop and discard it after the iteration than to store all $\binom{d^n}{k}$ k-subsets and traverse through the list. This is done using a k-subset unranking algorithm. Two of these algorithms (from [5]) are listed as Algorithms 2 and 3. These unranking functions allow us to completely parallelize the brute force algorithm, and the results obtained are listed in Figure 2.

Algorithm 2 Revolving Door Unranking Algorithm

```
procedure RevDoor($r, k, n$)                ▷ subset index, subset size, set size
    $x = n$
    for $i = k : 1$ do
        while $\binom{x}{i} > r$ do
            $x = x - 1$
        end while
        $t_i = x + 1$
        $r = \binom{x+1}{i} - r - 1$
    end for
end procedure
return $T = (t_1, t_2, \ldots, t_k)$
```

3 D-Wave Quantum Annealing Machine

Under the collaborative effort "Adiabatic Quantum Computing Applications Research" (14-RI-CRADA-02) between the Information Directorate and Lock-

Algorithm 3 Lexicographic Unranking Algorithm

```
1: procedure LexUnrank($r, k, n$)    ▷ subset index, subset size, set size
2:     $x = 1$
3:     for $i = 1 : k$ do
4:         while $r \geq \binom{n-x}{k-i}$ do
5:             $r = r - \binom{n-x}{k-i}$
6:             $x = x + 1$
7:         end while
8:         $t_i = x$
9:         $x = x + 1$
10:    end for
11: end procedure
12: return $T = (t_1, t_2, \ldots, t_k)$
```

$d \setminus n$	2	3	4	5
2	$\times$	4	6	12
3	4	9		
4	5			
5	6			

Figure 2: Results for $\mathcal{B}(d, n)$ obtained using HPC

heed Martin Corporation, we aim to extend the results obtained by the HPC system. In general, the D-Wave machine can address a class of Ising problems natively by the hardware. As stated in the D-Wave user documents, "The D-Wave hardware can be viewed as a hardware heuristic which minimizes Ising objective functions using a physically realized version of quantum annealing." [7] The Ising model is an energy minimization problem of -1/+1-valued variables. It can be converted to a quadratic unconstrained binary optimization (QUBO) problem that uses 0/1-valued variables, and so they are often used interchangeably.

3.1 Binary Optimization Model

We present a binary optimization formula for the minimum identifying code problem. Adjustments must be made to create a quadratic version. We will define this model using three separate functions: one to show that the set has the correct size, one to show that the set is dominating, and one to show that the set is separating (or identifying).

3.1.1 Variable Definitions

We will use the notation $B(v)$ for $v \in V(G)$, where $B(v) = N(v) \cup \{v\}$. In other words, $B(v)$ is the set containing all vertices adjacent to v, plus v itself. This is referred to in graph theory as the ball of radius one centered at v.

We define the variables as follows.

$$x_{vi} = \begin{cases} 1, & \text{if } i \in B(v); \\ 0, & \text{otherwise.} \end{cases}$$

3.1.2 Set S has size k

We define the first function, H_A, as follows.

$$\begin{aligned} H_A &= (k - \textstyle\sum_v x_{vv}) \\ &= 0 \qquad \text{iff} \quad |S| = k. \end{aligned}$$

3.1.3 Set S is a dominating set

By definition, this is equal to $\forall v \in G$, $B(v) \cap S \neq \emptyset$. This is equivalent to the following.

$$\begin{aligned} \forall v \in G, B(v) \cap S \neq \emptyset &\leftrightarrow (x_{vv} = 1) \vee \left(\sum_{uv \in E} x_{uv} \geq 1 \right) \\ &\leftrightarrow (1 - x_{vv} = 0) \vee \neg \left(\sum_{uv \in E} x_{uv} = 0 \right) \\ &\leftrightarrow (1 - x_{vv} = 0) \vee \left(\prod_{uv \in E} (1 - x_{uv}) = 0 \right) \end{aligned}$$

From this statement, we get the following equation for our second function.

$$H_B = \textstyle\sum_v (1 - x_{vv}) \cdot \left(\prod_{uv \in E} (1 - x_{uv}) \right)$$

3.1.4 Set S is a separating set

By definition, this is equal to $\forall x, y \in G$, $(B(x) \cap S) \triangle (B(y) \cap S) \neq \emptyset$. This is equivalent to the following for a specific pair $x \neq y$.

$$\begin{aligned} (B(x) \cap S) \triangle (B(y) \cap S) \neq \emptyset &\leftrightarrow \exists v \in (B(x) \cap S) \triangle (B(y) \cap S) \\ &\leftrightarrow \exists v, (v \in B(x) \cap S) \oplus (v \in B(y) \cap S) \\ &\leftrightarrow \exists v, (x_{xv} = 1) \oplus (x_{yv} = 1) \\ &\leftrightarrow \exists v, (1 - (x_{xv} + x_{yv}) = 0) \\ &\leftrightarrow \prod_v (1 - x_{xv} - x_{yv}) = 0 \end{aligned}$$

From this statement, we get the following equation for our third function, summed over all pairs x, y.

$$H_C = \textstyle\sum_x \sum_{y \neq x} \prod_v (1 - x_{xv} - x_{yv})^2$$

3.1.5 The Binary Optimization Model

From these three functions, our binary optimization model is the following.

$$\begin{aligned} H(S) &= H_A(S) + H_B(S) + H_C(S) \\ &= 0 \qquad \text{iff} \quad \text{S is an identifying code.} \end{aligned}$$

Note that while this does provide a binary optimization model for our problem, it is not quadratic. In order to convert $H(S)$ to a quadratic binary equation, each higher order term must be replaced with several new variables. While this is possible, it is a time-consuming and arduous process that introduces many new variables. Hence this approach will likely not be the most efficient implementation.

3.2 Integer Program Formulation

From [8], we have the following integer program formulation of the minimum identifying code problem in graphs.

First, we define the modified adjacency matrix as follows. It is the adjacency matrix plus the identity matrix.

$$A_{ij} = \begin{cases} 1, & \text{if } (i,j) \in E \text{ or } i = j; \\ 0, & \text{otherwise.} \end{cases}$$

Using this definition, we see that a ball of radius 1 centered at vertex i is given by the following vector.

$$B(i) = [A_{1i}, A_{2i}, \ldots, A_{ni}]^T$$

Our vertex subset S is defined as the following vector.

$$S = [s_1, s_2, \ldots, s_n]^T \text{ where } s_i = \begin{cases} 1, & \text{if } i \in S; \\ 0, & \text{otherwise.} \end{cases}$$

To compare two identifying sets with respect to S for vertices i and j, the following expression computes the size of $(B(i) \cap S) \triangle (B(j) \cap S)$.

$$\sum_{k=1}^{n} |A_{ki} - A_{kj}| \cdot s_k$$

This implies that in order for S to be a valid identifying code, we must have the following inequality satisfied for all pairs of vertices i and j.

$$\sum_{k=1}^{n} |A_{ki} - A_{kj}| \cdot s_k \geq 1$$

For the dominating property to be satisfied, we require the following additional inequality.

$$A \cdot S \geq \mathbf{1}^T$$

Thus our integer program is given by the following.

$$\begin{array}{lllll} \min & |S| & & & \\ \text{s.t.} & \sum_{k=1}^{n} |A_{ki} - A_{kj}| \cdot s_k & \geq & 1, & \forall i \neq j \\ & A \cdot S & \geq & \mathbf{1}^T & \\ & s_k \in \{0,1\} & & & \end{array}$$

In order to use these ideas for the D-Wave machine, our constraints must be equalities. This means we must add binary slack variables for each inequality. For the first set of inequalities, we must determine an upper bound for each inequality. Since these correspond to the constraint $|(B(i) \cap S) \triangle (B(j) \cap S)| \geq 1$, an easy upper bound is given by the following.

$$|B(i)| + |B(j)| \geq |(B(i) \cap S) \triangle (B(j) \cap S)| \geq 1$$

For the class of de Bruijn graphs, we are able to use this to get a bound on the number of slack variables needed. Since the maximum size of any ball in $\mathcal{B}(d,n)$ is $2d+1$, this gives us an upper bound of size $4d+2$ for this class of graphs. Hence for each inequality in this set, we must add $4d+2$ binary slack variables to convert the inequality to an equality. Since there are $\frac{d^n(d^n-1)}{2}$ possible pairs i, j, this implies that we must add a huge number of binary slack variables, equal to the following expression, just to satisfy the first set of inequalities.

$$d^n(d^n-1)(2d+1) \text{ slack variables}$$

Hence, this method is not going to be an efficient way to map our problem onto the D-Wave machine.

3.3 Satisfiability Formulation

This approach formulates the identifying code problem as a boolean satisfiability (SAT) problem. Each term in the SAT problem is mapped to an Ising model. The mapping introduces auxiliary variables so that the Ising model contains only quadratic and linear terms. A graph embedding technique is then used to map this Ising model onto the connectivity of the D-Wave chip. Finally, gauge transformations are used to mitigate the effects of intrinsic control errors. We will illustrate each step with an example of $\mathcal{B}(d,n)$ when $d=2$ and $n=3$. As stated previously, these methods easily apply to any graph.

3.3.1 SAT Formulation

First, we label the nodes of $\mathcal{B}(2,3)$ from 0 to $d^n - 1 = 7$. Then we define the set of variables $\{x_i\}$ for $i = 0, 1, \ldots, 7$ as follows.

$$x_i = \begin{cases} 1, & \text{if node } i \text{ is included in the identifying code;} \\ 0, & \text{otherwise.} \end{cases}$$

Next, we look at the ball for each node and form clauses corresponding to their domination constraints.

Ball	Contents	Constraints
$B(0) = B(000)$	$\{000, 001, 100\} = \{0, 1, 4\}$	$x_0 \vee x_1 \vee x_4$
$B(1) = B(001)$	$\{000, 001, 010, 011, 100\} = \{0, 1, 2, 3, 4\}$	$x_0 \vee x_1 \vee x_2 \vee x_3 \vee v_4$
$B(2) = B(010)$	$\{001, 010, 100, 101\} = \{1, 2, 4, 5\}$	$x_1 \vee x_2 \vee x_4 \vee x_5$
$B(3) = B(011)$	$\{001, 011, 101, 110, 111\} = \{1, 3, 5, 6, 7\}$	$x_1 \vee x_3 \vee x_5 \vee x_6 \vee x_7$
$B(4) = B(100)$	$\{000, 001, 010, 100, 110\} = \{0, 1, 2, 4, 6\}$	$x_0 \vee x_1 \vee x_2 \vee x_4 \vee x_6$
$B(5) = B(101)$	$\{010, 011, 101, 110\} = \{2, 3, 5, 6\}$	$x_2 \vee x_3 \vee x_5 \vee x_6$
$B(6) = B(110)$	$\{011, 100, 101, 110, 111\} = \{3, 4, 5, 6, 7\}$	$x_3 \vee x_4 \vee x_5 \vee x_6 \vee x_7$
$B(7) = B(111)$	$\{011, 110, 111\} = \{3, 6, 7\}$	$x_3 \vee x_6 \vee x_7$

From this set of constraints, we form clauses for each pairwise XOR of balls. This is shown in Figure 3

Now we can eliminate more specific clauses that are implied by more general clauses. For example, Figure 4 shows which two-term constraints imply the corresponding larger constraints. Hence, the only constraints that we have left are given below.

$$\begin{array}{l} x_2 \vee x_3 \\ x_0 \vee x_2 \vee x_5 \\ x_2 \vee x_6 \\ x_0 \vee x_3 \vee x_5 \\ x_3 \vee x_6 \\ x_0 \vee x_5 \vee x_6 \\ x_1 \vee x_2 \vee x_7 \\ x_1 \vee x_4 \\ x_1 \vee x_5 \\ x_2 \vee x_4 \vee x_7 \\ x_2 \vee x_5 \vee x_7 \\ x_4 \vee x_5 \end{array}$$

Satisfying this set of constraints are the four possible minimum solutions, given below.

$$\begin{array}{c} \{x_1, x_2, x_3, x_5\} \\ \{x_1, x_2, x_5, x_6\} \\ \{x_2, x_3, x_4, x_5\} \\ \{x_2, x_4, x_5, x_6\} \end{array}$$

3.3.2 Mapping SAT Clauses to Ising Models

We construct a Hamiltonian

$$\mathcal{H} = \sum_j \mathcal{H}_j(\{x_i, i \in A_j\}) + \lambda \sum_i x_i.$$

Each of the terms has the property that

$$x^* = \arg\min_{x_i} \mathcal{H}_j(\{x_i, i \in A_j\}) \text{ iff } \left(\bigvee_{i \in A_j} x_i \text{ is true} \right).$$

Figure 3: XOR balls for case $d = 2$, $n = 3$

	$B(1)$	$B(2)$	$B(3)$	$B(4)$	$B(5)$	$B(6)$	$B(7)$
$B(0)$	$x_2 \vee x_3$	$x_0 \vee x_2 \vee x_5$	$x_0 \vee x_3 \vee x_4 \vee x_5 \vee x_6 \vee x_7$	$x_2 \vee x_6$	$x_0 \vee x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5 \vee x_6$	$x_0 \vee x_1 \vee x_3 \vee x_5 \vee x_6 \vee x_7$	$x_0 \vee x_1 \vee x_3 \vee x_4 \vee x_6 \vee x_7$
$B(1)$		$x_0 \vee x_3 \vee x_5$	$x_0 \vee x_2 \vee x_4 \vee x_5 \vee x_6 \vee x_7$	$x_3 \vee x_6$	$x_0 \vee x_1 \vee x_4 \vee x_5 \vee x_6$	$x_0 \vee x_1 \vee x_2 \vee x_5 \vee x_6 \vee x_7$	$x_0 \vee x_1 \vee x_2 \vee x_4 \vee x_6 \vee x_7$
$B(2)$			$x_2 \vee x_3 \vee x_4 \vee x_6 \vee x_7$	$x_0 \vee x_5 \vee x_6$	$x_1 \vee x_3 \vee x_4 \vee x_6$	$x_1 \vee x_2 \vee x_3 \vee x_6 \vee x_7$	$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5 \vee x_6 \vee x_7$
$B(3)$				$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5 \vee x_7$	$x_1 \vee x_2 \vee x_7$	$x_1 \vee x_4$	$x_1 \vee x_5$
$B(4)$					$x_0 \vee x_1 \vee x_3 \vee x_4 \vee x_5$	$x_0 \vee x_1 \vee x_2 \vee x_3 \vee x_5 \vee x_7$	$x_0 \vee x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_7$
$B(5)$						$x_2 \vee x_4 \vee x_7$	$x_2 \vee x_5 \vee x_7$
$B(6)$							$x_4 \vee x_5$

Figure 4: Constraint Implications

General Constraint	Specific Constraints
$B(0) \oplus B(1) = x_2 \vee x_3$	$B(0) \oplus B(5)$, $B(2) \oplus B(3)$, $B(2) \oplus B(6)$, $B(3) \oplus B(4)$, $B(4) \oplus B(6)$, $B(4) \oplus B(7)$
$B(0) \oplus B(4) = x_2 \vee x_6$	$B(0) \oplus B(5)$, $B(1) \oplus B(3)$, $B(1) \oplus B(6)$, $B(1) \oplus B(7)$, $B(2) \oplus B(3)$, $B(2) \oplus B(6)$, $B(2) \oplus B(7)$
$B(1) \oplus B(4) = x_3 \vee x_6$	$B(0) \oplus B(3)$, $B(0) \oplus B(5)$, $B(0) \oplus B(6)$, $B(0) \oplus B(7)$, $B(2) \oplus B(3)$, $B(2) \oplus B(5)$, $B(2) \oplus B(6)$, $B(2) \oplus B(7)$
$B(3) \oplus B(6) = x_1 \vee x_4$	$B(0) \oplus B(5)$, $B(0) \oplus B(7)$, $B(1) \oplus B(5)$, $B(1) \oplus B(7)$, $B(2) \oplus B(5)$, $B(2) \oplus B(7)$, $B(3) \oplus B(4)$, $B(4) \oplus B(5)$, $B(4) \oplus B(7)$
$B(3) \oplus B(7) = x_1 \vee x_5$	$B(0) \oplus B(5)$, $B(0) \oplus B(6)$, $B(1) \oplus B(2)$, $B(1) \oplus B(5)$, $B(1) \oplus B(6)$, $B(2) \oplus B(7)$, $B(3) \oplus B(4)$, $B(4) \oplus B(5)$, $B(4) \oplus B(6)$
$B(6) \oplus B(7) = x_4 \vee x_5$	$B(0) \oplus B(3)$, $B(0) \oplus B(5)$, $B(1) \oplus B(3)$, $B(1) \oplus B(5)$, $B(2) \oplus B(7)$, $B(3) \oplus B(4)$, $B(4) \oplus B(5)$

We will show momentarily how the $\mathcal{H}_j$ are constructed. The last term $\lambda > 0$ is a penalty term that rewards smaller size codes. Therefore, the minimum solutions (or ground states) of $\mathcal{H}$ are the minimum identifying codes.

In order to solve the Hamiltonian using adiabatic quantum optimization, we have the further constraint that the $\mathcal{H}_j$ must contain only quadratic and linear terms in the binary variables $\{x_i\}$. In general to accomplish this, we must introduce auxiliary variables, which we will denote by $\{z_i\}$. Also, we will switch to the Ising model convention where each of the x_i and z_i can take values $\{+1, -1\}$ instead of $\{0, 1\}$.

The mapping from OR-clauses to Ising models that we will use, namely

$$\bigvee_{i \in A_j} x_i \mapsto \mathcal{H}_j(\{x_i, i \in A\}),$$

depends only on the number of variables $k = |A_j|$ in the OR-clause. These mappings for $k = 2$ through $k = 6$ are represented diagrammatically in Figure 5.

In the diagrams, numbers attached to a node represent the linear coefficients in the Ising model, while numbers attached to an edge represent the quadratic (coupling) coefficients in the Ising model. For example, the diagram for $k = 3$ represents the following Ising model.

$$\mathcal{H}_3(x_1, x_2, x_3, z_1) = x_1x_2 - 2x_1z_1 - 2x_2z_1 - 2x_2z_1 + z_1x_3 + x_1 + x_2 - 3z_1 - x_3$$

It can be confirmed that every ground state of $\mathcal{H}_3(x_1, x_2, x_3, z_1)$ satisfies $x_1 \vee x_2 \vee x_3$, and conversely every combination of $\{x_1, x_2, x_3\}$ that satisfies $x_1 \vee x_2 \vee x_3$ corresponds to a ground state of $\mathcal{H}_3(x_1, x_2, x_3, z_1)$.

3.3.3 Mapping the Ising model onto the D-Wave processor

In general, the graph of the Hamiltonian we get from the SAT-to-Ising mapping will *not* fit onto the D-Wave hardware graph. The D-Wave hardware graph, which is called the "Chimera" graph, is built up of unit cells, each of which is a four by four bipartite graph, $\mathcal{K}_{4,4}$. Even the simple Ising model for 3-OR shown in Figure 5 cannot be mapped directly onto the D-Wave hardware graph. This can be seen from the fact that our graph $\mathcal{B}(2, 3)$ contains a 3-cycle, whereas the smallest cycle possible on the D-Wave hardware graph is a 4-cycle.

Embedding

Our first step to embedding is to determine how to map our OR-clauses to the physical qubits. One way to embed the 3-OR graph onto the D-Wave is shown in Figure 6. We have mapped the logical qubit z_1 to two physical qubits, which are ferromagnetically coupled with a coupling strength -JFm.

Similarly, once we construct the full Ising Hamiltonian for the minimum identifying code problem, we can use embedding to map the graph of the Hamiltonian onto the D-Wave hardware graph.

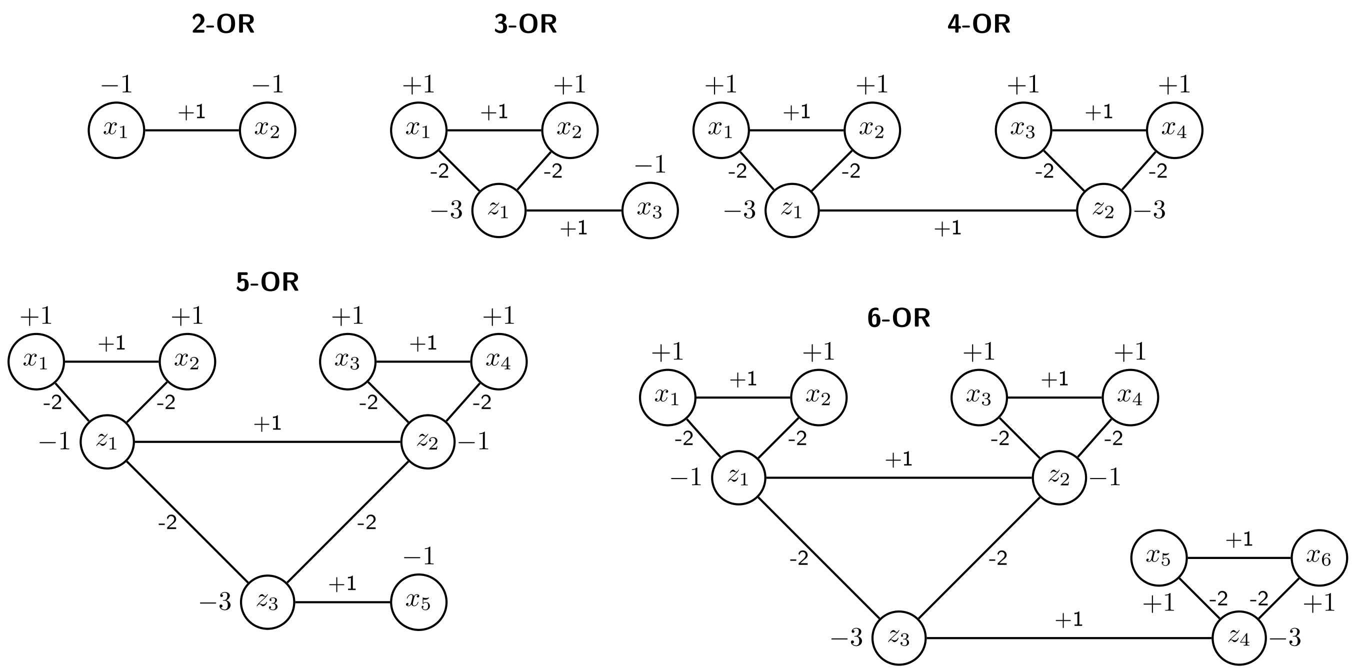

Figure 5: Mapping from OR-clauses to Ising Models

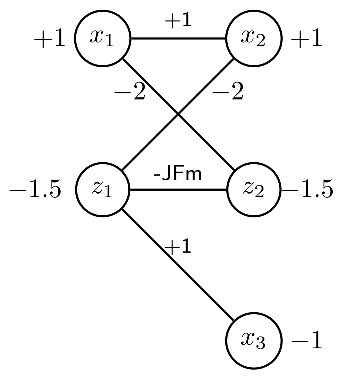

Figure 6: Embedding 3-OR onto the Chimera Graph

Problem Decomposition

If the graph of the Hamiltonian is too large to embed onto our current 504-qubit D-Wave hardware graph, one trick we can try is to decompose the SAT problem into smaller pieces that can be embedded. For example, in the $\mathcal{B}(2,3)$ example from earlier, one of the terms is $x_2 \vee x_3$. In order for this to be true, at least one of x_2 and x_3 must be true. We consider and solve each case separately.

If x_2 is true, then so is any OR-clause containing x_2, so we can eliminate those from the conjunction. We are left with the following clauses.

$$\begin{array}{l} x_3 \vee x_6 \\ x_0 \vee x_5 \vee x_6 \\ x_1 \vee x_4 \\ x_1 \vee x_5 \\ x_4 \vee x_5 \end{array}$$

Similarly, in the case where x_3 is true we can eliminate terms containing x_3, leaving the following clauses.

$$\begin{array}{l} x_2 \vee x_6 \\ x_0 \vee x_5 \vee x_6 \\ x_1 \vee x_2 \vee x_7 \\ x_1 \vee x_4 \\ x_1 \vee x_5 \\ x_2 \vee x_4 \vee x_7 \\ x_2 \vee x_5 \vee x_7 \\ x_4 \vee x_5 \end{array}$$

Both of these subproblems are simpler than the original problem and hence easier to embed. Whichever subproblem yields the smaller identifying codes will

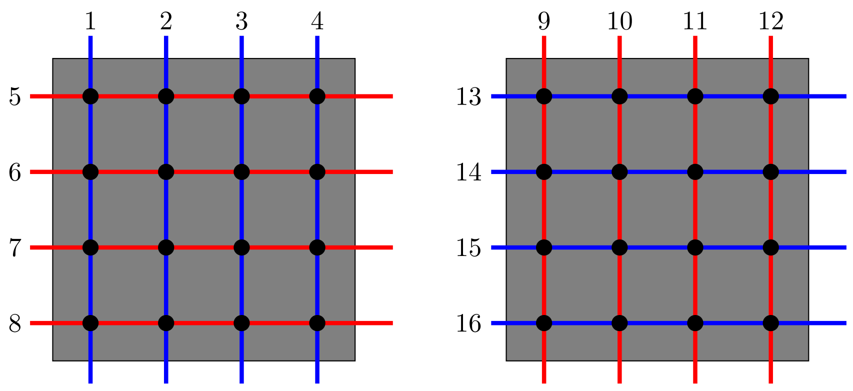

Figure 7: Example Gauge Transformation

be the solution of our original problem, or if both subproblems have minimum solutions of the same size, then we can take the union of the two solution sets.

Gauge Transformations

Embedding complex graphs leads to long *chains*, i.e. multiple physical qubits corresponding to the same logical qubit. In the current D-Wave embedding solver, all of the physical qubits in a chain will be ferromagnetically coupled with some coupling strength -JFm, which is iteratively increased to a large enough magnitude to ensure that all of the physical qubits in the chain agree (most of the time) and can be treated as a single logical qubit.

However, due to the characteristics of the D-Wave design, embeddings with many long ferromagnetically coupled chains will have a greater tendency for control precision errors which may affect solution quality. To combat these effects, we can utilize a *gauge transformation*, where we redefine a subset of the spin variables to be the opposite sign. Flipping a subset of the spins in this way induces a transformation on the Ising coefficients: If S_1 has been flipped ($S_1' = -S_1$), then $h_1' = -h_1$. If one of S_1 and S_2 has been flipped, then $J_{12}' = -J_{12}$. But if both of S_1 and S_2 have been flipped, then $J_{12}' = J_{12}$.

Consider for example the gauge transformation shown in Figure 7. In the figure, the red qubits are flipped by the gauge transformation while the blue qubits are unchanged. In the first unit cell, all of the horizontal qubits are flipped while the vertical qubits are unchanged. In the next unit cell, all of the vertical qubits are flipped while the horizontal qubits are unchanged. So, suppose our embedding contains the chain 1 (blue vertical qubit), 5 (red horizontal qubit), 13 (blue horizontal qubit). Note that each consecutive pair in the chain contains exactly one flipped qubit, so all of the ferromagnetic couplings -JFm in the chain will be replaced with antiferromagnetic couplings +JFm by the gauge transformation. By using gauge transformations like this, we may be able to reduce the control precision errors caused by embeddings with long chains.

3.3.4 Results for $\mathcal{B}(2, 4)$

By combining all of the above tricks, we obtained results for the minimum identifying code problem on the $d = 2, n = 4$ undirected de Bruijn graph on the D-Wave hardware.

SAT formulation

Figure 8 shows the full SAT formulation of the identifying code for $\mathcal{B}(2, 4)$. This was obtained by following the procedure outlined in section 3.3.1. It contains 50 clauses over 16 variables.

Problem Decomposition

For the full SAT formulation in Figure 8, the Ising model was too large to embed on the current D-Wave hardware graph. The problem must be decomposed further. Note that two of the terms are $x_3 \vee x_{11}$ and $x_4 \vee x_{12}$. Thus, at least one of x_3 and x_{11} must be true, and at least one of x_4 and x_{12} must be true. Considering the branch where x_3 and x_4 are true, the SAT problem reduces to the problem shown in Figure 9. This decomposed formulation consists of just 24 clauses over 14 variables.

SAT clause to Ising model mapping

Using the Ising model mappings shown in Figure 5, we generated an Ising model with 49 auxiliary variables $\{z_i\}$ for a total of 63 variables. We furthermore added the penalty term $\lambda \sum_i x_i$ so that the ground state will be a minimum identifying code. Note that this is far better than the 1200+ auxiliary (slack) variables required in the integer program formulation for this case.

Figure 10 shows the logical graph of the Ising model. In the figure, nodes corresponding to the original 14 boolean variables are shown in green; the remaining nodes represent the auxiliary variables added during the SAT-to-Ising mapping process.

Embedding

We used the D-Wave software function `sapiFindEmbedding()`, to find an embedding of this Ising model onto the hardware graph of the 504-qubit D-Wave machine at the USC-Lockheed Martin Quantum Computing Center in Marina del Rey, CA. Figure 11 shows the embedding that we used, consisting of 253 physical qubits, with a maximum chain length of 8.

In Figure 11, physical qubits corresponding to the same logical qubit have

```
( x2 v x3 )
( x0 v x2 v x4 v x5 v x8 v x9 )
( x0 v x3 v x6 v x7 v x8 v x9 )
( x0 v x1 v x2 v x4 v x9 v x10 )
( x4 v x12 )
( x0 v x1 v x6 v x9 v x12 v x14 )
( x0 v x3 v x4 v x5 v x8 v x9 )
( x0 v x2 v x6 v x7 v x8 v x9 )
( x0 v x1 v x3 v x4 v x9 v x10 )
( x1 v x5 v x8 v x10 )
( x1 v x4 v x9 v x10 v x11 )
( x0 v x2 v x5 v x8 v x9 v x12 )
( x1 v x3 v x5 v x12 )
( x1 v x2 v x9 v x10 v x13 )
( x1 v x6 v x9 v x11 v x14 v x15 )
( x1 v x3 v x7 v x8 v x12 v x14 )
( x1 v x3 v x6 v x9 v x14 v x15 )
( x4 v x5 v x8 v x9 v x11 )
( x0 v x1 v x2 v x9 v x10 v x12 )
( x3 v x8 v x10 v x12 )
( x2 v x5 v x8 v x9 v x13 )
( x2 v x4 v x11 v x13 )
( x2 v x6 v x7 v x10 v x13 )
( x2 v x5 v x6 v x13 v x14 )
( x3 v x5 v x7 v x12 )
( x3 v x10 v x12 v x14 )
( x3 v x5 v x6 v x13 v x14 v x15 )
( x3 v x6 v x7 v x10 v x13 v x15 )
( x3 v x11 )
( x0 v x1 v x4 v x6 v x9 v x14 )
( x4 v x6 v x7 v x10 v x11 )
( x4 v x5 v x6 v x11 v x14 )
( x5 v x7 v x10 v x14 )
( x5 v x6 v x11 v x12 v x14 v x15 )
( x5 v x6 v x11 v x13 v x14 v x15 )
( x6 v x7 v x8 v x9 v x13 v x15 )
( x6 v x7 v x8 v x9 v x12 v x15 )
( x6 v x7 v x10 v x11 v x12 v x15 )
( x6 v x7 v x10 v x11 v x13 v x15 )
( x12 v x13 )
( x0 v x1 v x8 )
( x1 v x2 v x4 v x5 v x9 )
( x1 v x3 v x6 v x7 v x9 )
( x2 v x4 v x8 v x9 v x10 )
( x2 v x5 v x10 v x11 )
( x4 v x5 v x10 v x13 )
( x5 v x6 v x7 v x11 v x13 )
( x6 v x8 v x9 v x12 v x14 )
( x6 v x10 v x11 v x13 v x14 )
( x7 v x14 v x15 )
```

Figure 8: SAT Formulation for $\mathcal{B}(2,4)$

```
( x0 v x1 v x6 v x9 v x12 v x14 )
( x0 v x2 v x6 v x7 v x8 v x9 )
( x1 v x5 v x8 v x10 )
( x0 v x2 v x5 v x8 v x9 v x12 )
( x1 v x2 v x9 v x10 v x13 )
( x1 v x6 v x9 v x11 v x14 v x15 )
( x0 v x1 v x2 v x9 v x10 v x12 )
( x2 v x5 v x8 v x9 v x13 )
( x2 v x6 v x7 v x10 v x13 )
( x2 v x5 v x6 v x13 v x14 )
( x5 v x7 v x10 v x14 )
( x5 v x6 v x11 v x12 v x14 v x15 )
( x5 v x6 v x11 v x13 v x14 v x15 )
( x6 v x7 v x8 v x9 v x13 v x15 )
( x6 v x7 v x8 v x9 v x12 v x15 )
( x6 v x7 v x10 v x11 v x12 v x15 )
( x6 v x7 v x10 v x11 v x13 v x15 )
( x12 v x13 )
( x0 v x1 v x8 )
( x2 v x5 v x10 v x11 )
( x5 v x6 v x7 v x11 v x13 )
( x6 v x8 v x9 v x12 v x14 )
( x6 v x10 v x11 v x13 v x14 )
( x7 v x14 v x15 )
```

Figure 9: Decomposed SAT formulation for x_3, x_4 true.

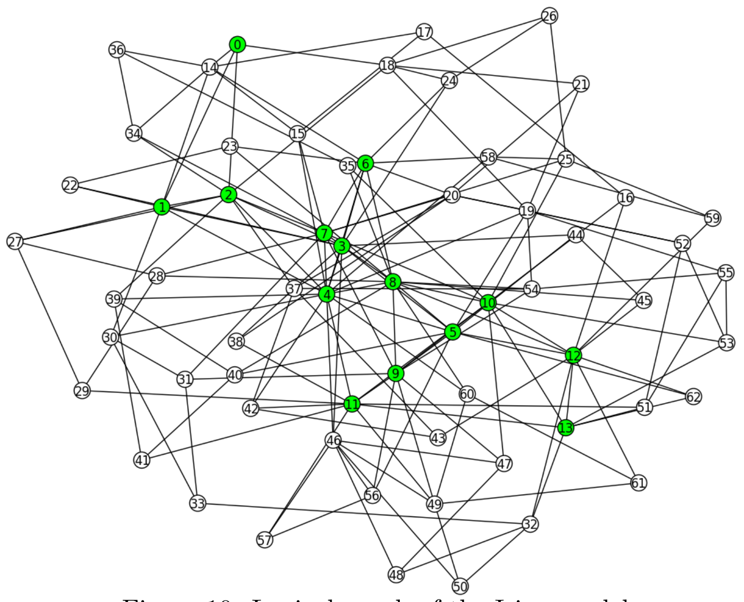

Figure 10: Logical graph of the Ising model

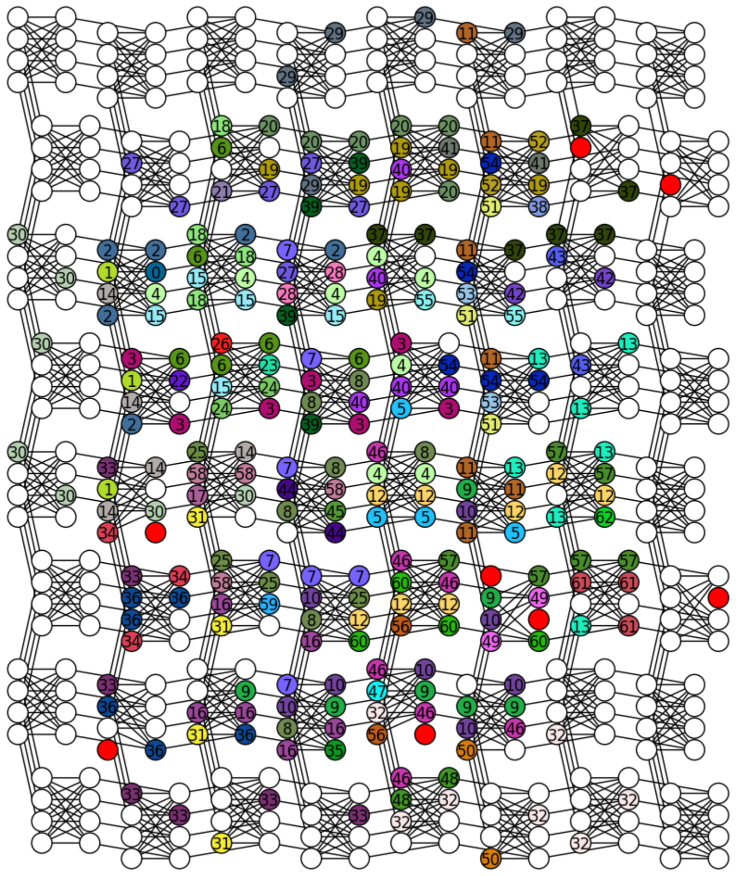

Figure 11: Embedding the problem onto the D-Wave hardware

the same color and are labeled with the same number. The unlabeled red qubits are known faulty qubits and are not used.

Gauge Transformations

Since we have an exact solution from the HPC system on the graph $\mathcal{B}(2,4)$, we know that the $x_3 = x_4 = 1$ branch of the problem should have a ground state with the remaining variables

$$x_8 = x_{10} = x_{13} = x_{14} = 1$$

corresponding to the minimum code size of 6. However, using the standard D-Wave embedding solver (which does not yet support gauge transformations - it uses all ferromagnetic couplings for the chains), we were not able to find this ground state. Even using the maximum allowed combinations of D-Wave function parameters `annealing_time` and `num_reads` (the number of annealing runs per call), the best that we could obtain were solutions corresponding to a code size of 7.

On the other hand, using a locally developed embedding solver, which also has the capability to incorporate gauge transformations, we were able to find the above ground state corresponding to a code of size 6, using the gauge transformation shown in Figure 7.

3.3.5 Scaling for Larger Cases

While the $\mathcal{B}(2,4)$ case is the largest that can be solved on the current 504-qubit D-Wave processor, we can make some rough estimates of how the qubit resource requirements scale for larger cases. For binary ($d = 2$) de Bruijn graphs, we extended the techniques described above to generate Ising models for the minimum identifying code problem for the cases $n = 3$ through $n = 6$. These models were then embedded into ideal Chimera graphs of various sizes using the D-Wave software function `sapiFindEmbedding()`. Figure 12 shows how the number of qubits needed for the embedding grows as a function of n.

Since the embedding algorithm is heuristic, it is possible that smaller embeddings could be found, so the embedding sizes shown here should be viewed as upper bounds. We computed embeddings firstly for larger versions of the current C8 Chimera architecture, which has 8-qubit unit cells, as well as for hypothetical C16 (16-qubit unit cell) Chimera graphs. For the C8 graphs, we started with the current Vesuvius architecture, which contains 64 8-qubit unit cells arranged in an 8x8 square grid, and increased the grid size to 12x12, 16x16, 24x24, 32x32, and so on until the embedding was successful. For the C16 graphs, we started with an 8x8 grid of 16-qubit unit cells, and increased the grid size to 16x16,24x24, and 32x32, until the embedding was successful. The plots show that for any given case, the embedding size on C16 is smaller than on C8 due to the higher graph connectivity of the C16 architecture; in other words, qubit resource requirements depend on the hardware graph connectivity.

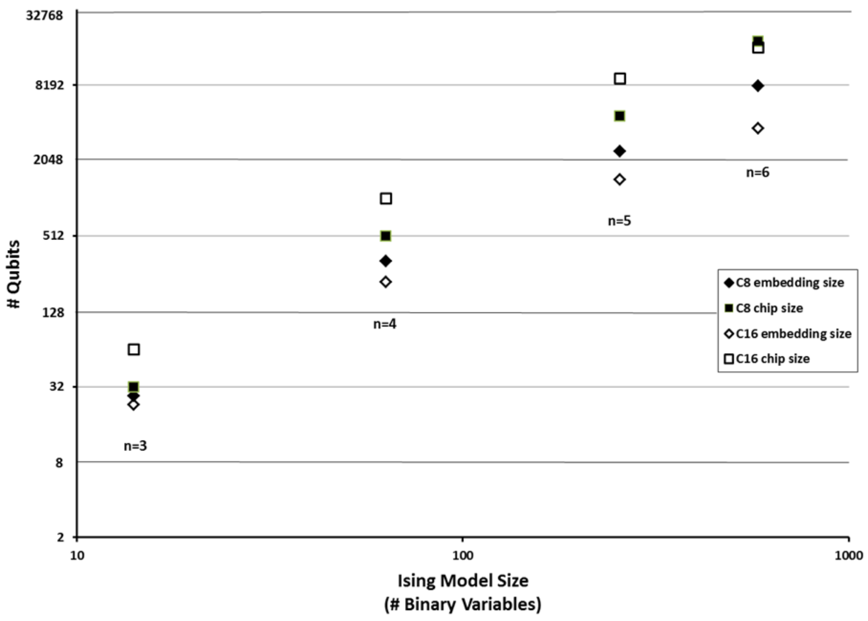

Figure 12: Estimated scaling of qubit resource requirements for minimum identifying code problem on binary de Bruijn graphs

From the figure, we can project roughly when the D-Wave processor would have sufficient qubits to accommodate the larger cases of the minimum identifying code problem for undirected binary de Bruijn graphs. If we define a processor generation to be 4 times the number of qubits as the previous generation (e.g. the “Vesuvius” generation had a 512-qubit design whereas the prior “Rainier” generation had a 128-qubit design), we can state that in roughly 1.5 generations we would be able to fit the $\mathcal{B}(2,5)$ case on the processor. This is the largest $d=2$ case for which the minimum identifying code size is presently known. In one more generation beyond that, we would be able to fit the $\mathcal{B}(2,6)$ case on the processor, for which the minimum identifying code size is presently not known.

Whether these hypothetical D-Wave processors would actually be able to solve these larger cases, will depend on the performance characteristics of those machines which is yet to be demonstrated. The hardware intrinsic control errors would need to be significantly reduced from the current levels. For example, the embedding we found for the $\mathcal{B}(2,6)$ case on the C8 architecture had a maximum chain length of 63 qubits. Solving a problem with chain lengths this large would probably require 6 to 7 bits of precision (the current Vesuvius design has 4 bits of precision). On the C16 architecture, the embeddings are less complex; e.g. the embedding we found for the $\mathcal{B}(2,6)$ case only had a maximum chain length of 25 qubits. However, it is not clear how difficult it would be for D-Wave to achieve a 16-qubit unit cell design.

Due to risk factors such as these, it could take longer than the projected number of processor generations before a solution can be found to the $\mathcal{B}(2,6)$ case. On the other hand, it may be possible to break the problem down into subproblems using the decomposition technique described above, which may make it easier to fit the problem onto the D-Wave processor.

4 SMT Solvers

Satisfiability Modulo Theory (SMT) is a current area of research that is concerned with the satisfiability of formulas with respect to some background theory [1]. SMT solvers combine boolean SAT solving with decision procedures for specific theories. For example, consider the following problem.

$$a = b + 1, \quad c < a, \quad c > b$$

In the theory of the integers, this problem is not satisfiable (there are no integers a, b, c where all the expressions are true), however in the theory of the real numbers it is satisfiable (for example, with a=11, b=10, c=10.5). In general, solving an SMT problem consists of first solving a SAT problem, then doing theory-specific reasoning, and then possibly going back and changing the SAT problem. This process is repeated if necessary. In addition, multiple theories can also be used in the same SMT problem instance, which may require additional repeats of this method.

$d \setminus n$	2	3	4	5	6	7
2	$\times$	4	6	12	(24)	(110)
3	4	9				
4	5	15				
5	6					
6	8					
7	9					
8	(10)					

Figure 13: Minimum identifying codes on $\mathcal{B}(d, n)$

To use SMT solvers on our identifying code problem for the undirected de Bruijn graph, we must first come up with a formulation of the problem using decision procedures. The graph $\mathcal{B}(d, n)$, contains d^n nodes. For each of these, we create a boolean variable that denotes whether or not the node is part of the identifying code. We then also create an array of boolean variables for that node's identifying set. An assertion is added to make sure that each element of the array is true if and only if the corresponding neighbor's boolean variable is true (i.e. if and only if the neighbor is part of the identifying code). To ensure unique codes, we add a statement to require that each node's identifying set is unique from every other node's identifying set. Then, to get codes of a fixed size, we create an integer variable for each node and add the constraints that the integer is at least 0 and no greater than 1. Next we add an assertion that each node's integer variable is 1 if and only if its boolean variable is true. Finally, we add a constraint that the sum of all of the integer variables is equal to the desired identifying code size.

Now that the formulation of the problem has been determined, we can use a commercial SMT solver to find solutions. For this work, we used the solver Z3, made by Microsoft Research. We begin by first picking a code size, and asking if there exists an identifying code of that size. If not, then the code size is increased by 1 and the problem is posed to Z3 again. This continues until an identifying code of a specific size is found. To find *all* satisfying models, after a single model was found an assertion is inserted into the formulation that requires that the the previously found identifying code be eliminated as an option. This forces Z3 to produce a different solution, or to state that the formulation is unsatisfiable (and hence no more identifying codes of that size exist). This process is repeated in a loop to obtain all identifying codes.

Using this approach on a single core, we were able to reproduce our results for $\mathcal{B}(d, n)$ from the HPC method in much less time. See Figure 13 for a summary of these results. The numbers in parentheses denote that we found a code of that size, but did not eliminate the possibility of a smaller code existing.

Because of the advancements in current SAT and SMT solvers, they offer the potential to scale much better than a parallelized brute force approach. This is due in part to the fact that many of today's solvers are capable of realizing which subsets of assignments will define an unsatisfiable result, and hence they will avoid models in which those statements are set. In our problem, this might correspond to a case in which nodes A and B have the same identifying set. In this case, the solver would not bother looking at combinations of True/False assignments on the other nodes that do not affect the identifying sets of A or B.

In addition to the sophistication of today's solvers, there is also the possibility of parallelizing the search. While some instances were run manually in a parallel manner for this experiment, there is some research to be done on automatically parallelizing the search in order to further our known minimum results.

5 Conclusions

The various methods discussed provide pure mathematicians with a range of opportunities for collaboration with scientists from several disciplines, such as computer science and physics. While it has been shown that some of these ideas correspond well to other combinatorial-type problems (for example, see [6] for a discussion of Ising models for problems from graph theory), we found that these approaches alone did not provide a strong enough method for this problem. Additionally, the relatively recent appearance of the D-Wave machine allowed for very little literature to draw from. From the base cases that we constructed using our variety of approaches, several new conjectures have been developed and eventually proven true [2], however the leg work needed to compute the base cases required a deep understanding of various computing techniques.

Acknowledgments

Work using the D-Wave quantum annealing machine was performed jointly by AFRL/RI and Lockheed Martin under Air Force Cooperative Research and Development Agreement 14-RI-CRADA-02.

S. Adachi was supported by Internal Research and Development funding from Lockheed Martin. S. Adachi would also like to thank Todd Belote and Dr. Andy Dunn of Lockheed Martin for their assistance respectively with the SAT-to-Ising mapping in Section 3.3.2, and with the generation of models for the scaling analysis of larger cases in Section 3.3.5.

LOCKHEED MARTIN and LOCKHEED are registered trademarks in the U.S. Patent and Trademark Office owned by Lockheed Martin Corporation.

References

[1] C. Barrett, R. Sebastiani, S.A. Seshia, and C. Tinelli, "Satisfiability Modulo Theories", in Handbook of Satisfiability, Chapter 26 (2009), 825-885.

[2] D. Boutin and V. Horan, “Identifying Codes on Directed De Bruijn Graphs”, submitted.

[3] I. Charon, O. Hudry, and A. Lobstein, “Minimizing the Size of an Identifying or Locating-Dominating Code in a Graph is NP-Hard”, *Theoret. Comput. Sci.*, **290** (2003) no. 3, 2109-2120.

[4] M.G. Karpovsky, K. Chakrabarty, and L.B. Levitin, “On a New Class of Codes for Identifying Vertices Graphs”, *IEEE Trans. Inf. Theory*, **355** (1998) no. 2, 599-611.

[5] Donald L. Kreher and Douglas R. Stinson, “Combinatorial Algorithms: Generation, Enumerations, and Search”, in Discrete Mathematics and Its Applications, Book 7, CRC Press (1998): Boca Raton.

[6] A. Lucas, “Ising Formulations of Many NP Problems”, *Frontiers in Physics*, **2**:5 (12 Feb 2014).

[7] “Programming with QUBOs,” (instructional document) D-Wave: The Quantum Computing Company, 2013.

[8] Y.C. Xu and R.B. Xiao, “Identifying Code for Directed Graph,” in Software Engineering, Artifical Intelligence, Networking, and Parallel/Distributed Computing, 2007.